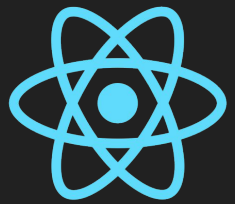
Vuejs

Les frameworks Javascript

- Essor du javascript et des frameworks js depuis le web 2.0, en remplacement de flash.
- Début d'une nouvelle ère de framework javascript avec l'apparition d'Angular en 2009.
- Ces outils permettent de créer plus rapidement une SPA en permettant aux développeurs de se concentrer sur l'aspect métier.



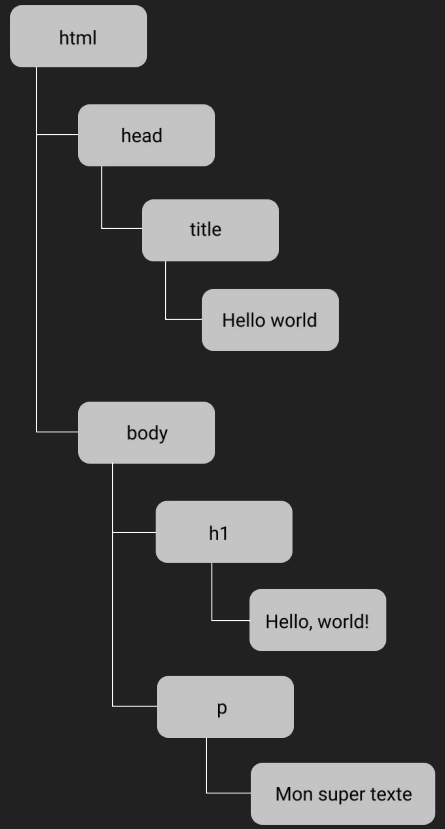
C’est quoi une SPA ?

- Une SPA (Single Page Application) est une application accessible par une page unique sur un site web.
- Les SPA permettent de rendre l’expérience utilisateur beaucoup plus dynamique et plus riche.
- Plus compliquée à prendre en main que du simple HTML ou une app monolithique
- Pas forcément compatible avec nos amis les browsers Microsoft.

Le DOM

Le DOM (Document Object Model) est une représentation d'un document HTML sous forme d'un arbre d'objet.

```html
<!doctype html>
<html lang="fr">
 <head>
   <title>Hello world</title>
  </head>
 <body>
    <h1>Hello, world!</h1>
    <p>Mon super texte</p>
  </body>
</html>
```



Shadow DOM

On peut considérer le shadow DOM comme un «DOM dans le DOM». Il possède une arborescence DOM isolée avec ses propres éléments et styles, complètement isolée du DOM d'origine. Le shadow DOM a été utilisé pendant des années pour créer et styliser des composants complexes tels que des éléments de formulaire.

Par exemple l'élément range : **<input type="range">**

Virtual DOM

Le DOM virtuel a été créé pour résoudre ces problèmes de nécessité de mettre à jour fréquemment le DOM d'une manière plus performante.

Un DOM virtuel peut être considéré comme une copie du DOM d'origine. Cette copie peut être fréquemment manipulée et mise à jour, sans utiliser les API DOM. Une fois que toutes les mises à jour ont été apportées au DOM virtuel, les modifications sont appliquées sur le DOM de manière ciblée et optimisée.

```
{
    tagName: "html",
    children: [
        {
            tagName: "head",
            children: [
              {
                tagName: "title",
                textContent: "Hello world",
              }
            ],
        },
        {
            tagName: "body",
            children: [
                {
                    tagName: "h1",
                    textContent: "Hello, world!"
                },
                {
                    tagName: "p",
                    textContent: "Mon super texte"
                },
            ]
        }
    ]
}
```

Et donc Vuejs ?

- Framework javascript permettant de créer des SPA
- Créé en 2014 par Evan You
- Inspiré par AngularJS, Vuejs reprend les meilleurs fonctionnalités tout en étant beaucoup plus léger que Angular.
- Courbe d’apprentissage beaucoup plus rapide que React ou Angular

Partie I, les bases

Installation

Vue propose 3 modes d’importation de la librairie dans une application web :

- CDN
- Npm
- CLI

Installation : CDN

On peut importer directement Vue dans notre page html :

```
<!-- Importation de la version de dev -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

<!-- Importation de la version de prod -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

Installation : npm

On peut importer directement Vue dans notre page html :

```
# On ajoute siplement vue à notre projet javascript
npm install vue
```

Installation : CLI

On installe le cli en global sur sa machine

```
npm install -g @vue/cli @vue/cli-service-global
# or
yarn global add @vue/cli @vue/cli-service-global
```

Installation : CLI

On peut ensuite créer un projet

```
vue create mon-super-projet
```

Les components

Il est recommandé de séparer son code en plusieurs components pour une question de lisibilité et de séparation des préocupations.

```
<template>

    Votre component

<template>

<script>

    Les méthodes, le state, les computed etc.

</script>

<style>

    Le style de votre component

</style>
```

Les components

Le component doit toujours être entouré par un component root, ici

```
<template>
  <div>
    <h1> {{ message }} </h1>
  </div>
</template>


<script>
export default {
  data(){
    return {
        message: "Hello !",
    };
  }
}
</script>
```

On utilise les “moustaches” pour afficher une variable dans le component

Toute la logique du component doit être à l’intérieur du export default

Le state du component est défini par le retour de la fonction data.

State

Le state (aussi appelé état dans nos contrées) est ce qui permet de rendre nos components dynamique. Dans Vuejs c'est le retour de la fonction "data" qui défini le state du component.

```
<script>
export default {
    name: "navbar",
    data() {
        return {
            isOpen: false
        };
    }
};
</script>
```

Props

Les props sont des variables qui sont passé dans l'appel du component. On peut lister nos props en array ou alors en objet. Lister nos props en objets permet de préciser leur type, et donc de mieux gérer les erreurs.

```
<script>
export default {
    name: "navbar",
    props: ['title', 'name']
};
</script>
```

```
<script>
export default {
    name: "navbar",
    props: {
      'title': String,
      'name': String
    }
};
</script>
```

```
<navbar title="foo" name="bar" />
```

Bind

On a déjà vu les “moustaches” qui permettent d’afficher une variable directement dans le component. Il existe aussi le v-bind qui permet d’utiliser une variable dans un élément html

```
<navbar v-bind:title="foo" name="bar" />

// Ou on peut utiliser le shorthand

<navbar :title="foo" name="bar" />
```

Conditions

Vous pouvez afficher ou masquer de manière dynamique les éléments de votre component grâce à **v-if**. Vous pouvez utiliser **v-else** de la même manière qu'un else.

```
<template>
  <div>
    <p v-if="sayHello"> Hey </p>
    <p v-else> Hello </p>
  </div>
</template>
```

Conditions

Vous pouvez utiliser **v-if** directement sur le template de votre component !

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

Conditions

Vous pouvez aussi utiliser **v-show**. La différence de v-show est qu'il va render l'élément dans le DOM. Si la condition dans **v-show est** à true, l'élément vera sa propriété css "display" permutée.

```
<h1 v-show="ok">Hey !</h1>
```

Itérations

Vue vous permet aussi d'itérer sur des variables de votre state. Dans cet exemple on itère sur le tableau items. Il est très fortement recommandé d'utiliser ":key" sur votre élément pour permettre à Vue de garder la trace de chaque noeud.

```
<template>
  <ul>
    <li v-for="item in items" :key="item.message">
      {{ item.message }}
    </li>
  </ul>
</template>

<script>
export default {
  data() {
    return {
      items: [
        { message: 'Foo' },
        { message: 'Bar' }
      ]
    };
  }
}
</script>
```

Itérations

Vous pouvez aussi itérer sur des objets.

```vue
<template>
  <ul>
    <li v-for="value in object">
      {{ value }}
    </li>
  </ul>
</template>

<script>
export default {
  data() {
    return {
      object: {
        title: 'How to do lists in Vue',
        author: 'Jane Doe',
        publishedAt: '2016-04-10'
      }
    };
  }
}
</script>
```

Itérations

Et vous pouvez utiliser autant de propriété que vous voulez !

```
<div v-for="(value, name) in object">
  {{ name }}: {{ value }}
</div>
...
<div v-for="(value, firstname, lastname) in object">
  {{ firstname }} {{ lastname }}: {{ value }}
</div>
```

Événements

Vue supporte tous les events présents nativement dans le DOM. On peut par exemple attacher un event click sur un bouton.

```
<button v-on:click="console.log('clicked')"> Click me ! </button>

// Vous pouvez utiliser le shorthand @

<button @click="console.log('clicked')"> Click me ! </button>
```

Méthodes

Lorsque vous écrivez votre component, vous trouverez utile de pouvoir écrire des fonctions pour les utiliser dans le component.

```
<template>
  <button @click="sayHi"> Click me to say hi ! </button>
</template>

<script>
  export default {
    methods: {
      sayHi() {
        alert('hi !');
      },
    }
  }
</script>
```

Computed

Les computed (ou variables calculées) vous permettent …. d'utiliser des variables calculées dans votre component.

```
<template>
  <div>
    <p> Your name is {{ fullname }} </p>
  </div>
</template>

<script>
  export default {
    data() {
      return {
        lastname: "Durand",
        firstname: "Kevin",
      }
    },
    computed: {
      fullname() {
        return this.firstname+' '+this.lastname;
      },
    }
  }
</script>
```

Slots

Les slots vous permettent de passer des élements enfants à vos components.
Tous les éléments/composants qui seront à l'intérieur de votre component pourront être rendu dans votre component grâce à **<slot />**

```
<my-component>
  Hey
</my-component>

// Et dans votre component
<template>
    <div>
        <h1>
          <slot></slot>
        </h1>
    </div>
<template>
```

V-model

v-model vous permet de créer une liaison de données bidirectionnelle sur les champs de formulaire.

```
<input v-model="message" placeholder="modifiez-moi">
<p>Le message est : {{ message }}</p>
```

Exercices

Exercice 1 :

Créez un component **<id-card>** qui affiche un avatar, un nom, un prénom et une date de naissance. Les données affichés devront être passée en props. Le nom et le prénom (fullname) doivent être une variable calculée.

Exercice 2 :

Créez un deuxième composant qui affiche les <id-card /> en liste. En haut de la liste, créez un bouton qui permet de trier les cards par date de naissance.

Exercices

Exercice 3 :

Créez un formulaire qui permet d'ajouter une nouvelle **id-card** dans la liste

Exercice 4 :

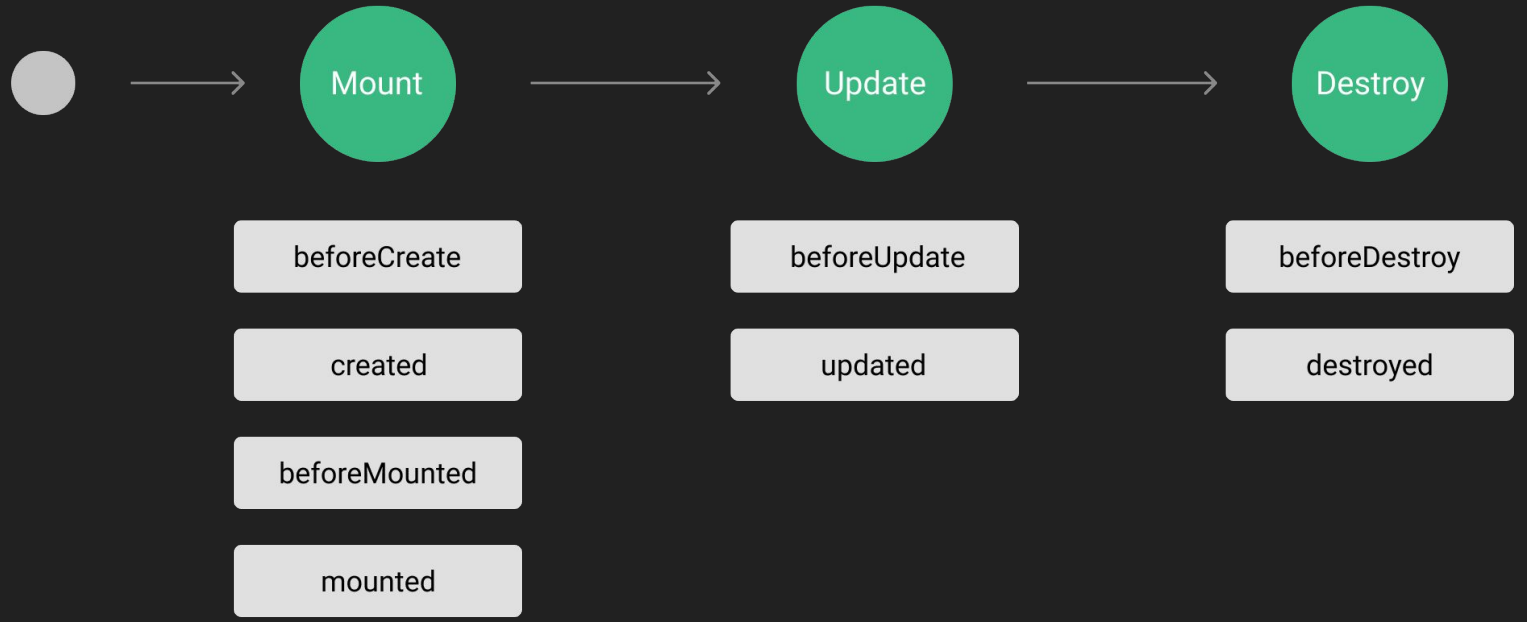
Rajoutez un bouton à **id-card** qui permet de la supprimer

Exercice 5 :

Rajoutez une deuxième bouton à **id-card** qui permet d'éditer les champs

Partie II, utilisation avancée

Lifecycle de Vuejs



Router

```
// router/index.js

import Router from 'vue-router';
import Vue from 'vue';
import CustomComponent from '@/component/CustomComponent';

Vue.use(Router);

export const router = new Router({
  mode: 'history',
  base: process.env.BASE_URL,
  routes: [
    {
      component: CustomComponent,
      path: '/',
    },
  ]
});
```

Router

```
// main.js

import Vue from 'vue';
import router from '@/routes';
import App from '@/App.vue';

new Vue({
  router,
  render: h => h(App)
}).$mount('#app');
```

```
// App.vue

<template>
  <div id="app">
      <router-view />
  </div>
</template>

<script>
export default {
  name: 'App'
};
</script>
```

Router

```
<div id="app">
  <p>
    <!-- router-link vous permet de créer des liens dans le router -->
    <router-link to="/home">Home</router-link>
    <router-link to="/abot">About</router-link>
  </p>
  <!-- router-view va render le component qui correspond à la route actuelle -->
  <router-view></router-view>
</div>
```

Router

Le router de Vue vous permet de gérer des paramètres d'url. Dans notre component on peut récupérer le paramètre de route id grâce à ***$route.params.id***

```
new VueRouter({
  routes: [
    { path: '/user/:id', component: User }
  ]
})
```

Router

Vous pouvez push une route de manière programmatique dans votre component grâce à ***router.push***.

```
// literal string path
router.push('home')

// object
router.push({ path: 'home' })

// named route
router.push({ name: 'user', params: { userId: '123' } })

// with query, resulting in /register?plan=private
router.push({ path: 'register', query: { plan: 'private' } })
```

Utilisation avec une api

On vient de découvrir en grande partie les fonctionnalités de Vue. On peut donc désormais construire un site statique comme un site vitrine en Vue. Heureusement on ne va pas s'arrêter là. On va voir comment interfacer une API avec Vuejs. Pour cela, on va utiliser axios qui permet de faire des calls HTTP.

```
npm install axios
```

Utilisation avec une api

Axios permet d'utiliser une grande partie des verbes HTTP (get, post, put, patch) pour faire vos requêtes. Dans cet exemple on fait un get sur la SWAPI et on stock le résultat dans notre state. On peut ensuite l'afficher grâce à **v-for.**

```
<template>
<div>
  <ul>
    <p v-if="error"> {{ error }} </p>
    <li v-for="item in items">
      {{ item.name }}
    </li>
  </ul>
</div>
</template>

<script>
import axios from 'axios';

export default {
  name: 'example',
  data() {
    return {
      items: [],
      error: null,
    };
  },
  mounted() {
    axios.get('https://swapi.dev/api/people/')
      .then(res => {
        this.items = response.data.results;
      })
      .catch(err => {
        this.error = `Error (${err.response.status})`;
      });
  },
}
</script>
```

Exercices

Exercice 1 :

Reprenez votre component **id-card.** En utilisant la **SWAPI**, affichez les informations des personnages de star wars dans **id-card.**

Exercice 2 :

Grâce au router, créer des onglets pour les endpoints People, Starships, Vehicles, Species, Planets. Affichez une liste de component **id-card** correspondant à chaque onglet.

Authentification en vuejs

Authentification par jeton

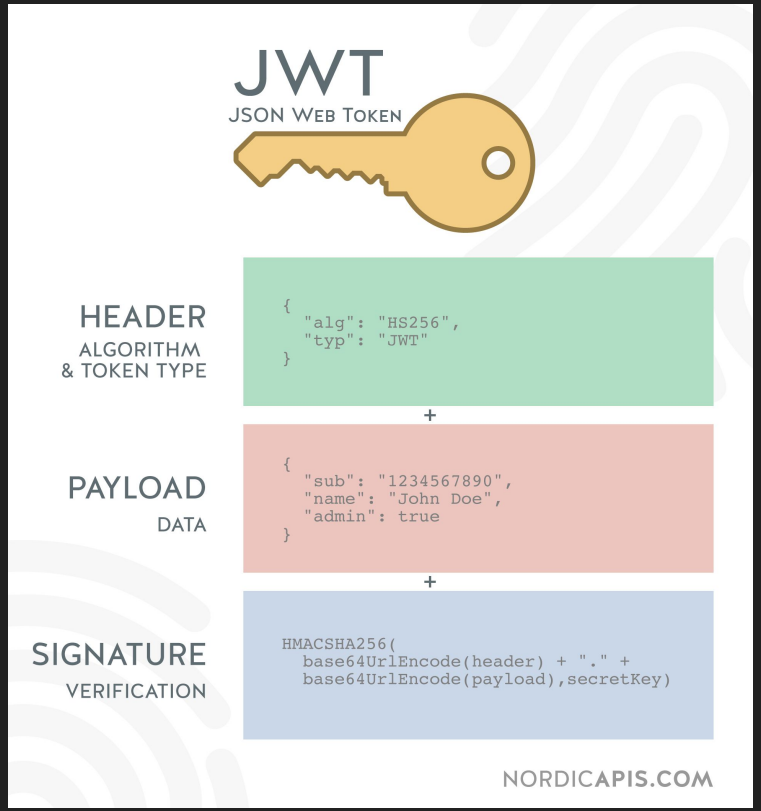
Avec l'authentification par Token, le serveur crée un JSON Web Token (JWT) et envoie le jeton au client.

Le JWT est généralement stocké dans un stockage local ou dans le cookie, et il est inclus dans chaque demande faite par l'utilisateur.

Le serveur validera le JWT. L'authentification par JWT est une approche plus moderne utilisée dans les nouvelles applications Web et pour les appareils mobiles. L'état de l'utilisateur n'est pas stocké sur le serveur, il est stocké dans le jeton.

JWT

Le JSON Web Token permet d'échanger des informations entre deux services (par exemple une api back et un front) de manière sécurisés. Les jetons sont chiffrés grâce à une clef secrète qui est vérifiée par l'émetteur (généralement l'api) qui permet la sécurisation des données.



Projet

Vous devez créer le front d’un site de news sur la musique.

Sur ce site on peut y trouver des news, les prochains concerts d’artistes et de groupe. Chaque artiste/groupe a une page dédiée où on peut voir ses albums et ses concerts. Sur la page artiste il y a une aussi une photo et une description. Côté back office, les utilisateurs admin pourront modifier les news, les artistes, les concerts et les albums. La page d'accueil du site affichera les dernières news et albums.

Bonus :

- Les internautes peuvent s’inscrire pour commenter les news et mettre des likes sur les artistes. La page d'accueil le top 5 des artistes les plus likés.
- Gérer la recherche
- Gérer la pagination
- Utiliser ElementUI, Tailwind ou autre lib UI
- Utiliser Nuxtjs
- Utiliser Vuex

Projet : rendu

Vous devez rendre le projet avant le 6 juin à 23:00.

Pour le rendu :

- Par email à **cours@narah.io** avec dans l'objet du mail [ECV]. Dans votre email envoyez bien le lien du repo et le nom des personnes de votre groupe. Pour m'ajouter sur le repo privée : github & gitlab username : **louishrg**